Multitasking met de ESP32 (4)

binaire semaforen

Warren Gay (Canada)

Bij multitasking in FreeRTOS is het vaak nodig om tussen taken te synchroniseren. Dat is onder andere mogelijk met een binaire semafoor. Hier bekijken we de binaire semafoor en illustreren het gebruik met een eenvoudig programma.

Wat is een semafoor?

Een semafoor is een functie waarmee de aanroeper de uitvoering ervan kan blokkeren totdat toestemming is gegeven om verder te gaan. Met andere woorden, de aanroeper probeert de semafoor te 'nemen', maar als de semafoor al bezet is, wacht de aanroepende taak (blokkeert deze).
Zo kunnen we een van de eigenschappen van een semafoor definiëren: een binaire semafoor kent twee toestanden:

- bezet (*taken*)
- vrij (*given*)

Hoe biedt een semafoor bescherming?

Een semafoor beheert de toegang tot een bron niet zelf. De bescherming van systeembronnen kan alleen via een semafoor worden geïmplementeerd *als dat zo is afgesproken*. De semafoor werkt volgens een protocol: de *accessor* stemt ermee in om de beschikbare systeembronnen alleen te gebruiken als hij erin slaagt de semafoor te nemen. Ook zal hij de systeembron niet opnieuw gebruiken totdat hij de volgende vrije semafoor in bezit heeft genomen.

Time-outs bij het nemen

Met binaire semaforen kan de aanroeper een time-out periode specificeren in (systeem-ticks). Als een poging om een semafoor te nemen langer duurt dan het gespecificeerde aantal ticks, retourneert de aanroep een foutcode. Als alternatief kan FreeRTOS ook worden opgedragen om voor onbepaalde tijd te wachten tot de semafoor wordt vrijgegeven (net als een hopeloos verliefde jongen). Ten slotte is er de optie om helemaal geen time-out te specificeren, waarbij de aanroep meteen mislukt als de semafoor niet kan worden genomen. Samenvattend kunnen semaforen op drie manieren met de tijd omgaan:

- blokkeer continu tot de semafoor door de aanroeper kan worden 'genomen';
- blokkeer gedurende een bepaalde periode, de aanroep mislukt na verstrijken van de time-out;.
- de aanroep mislukt onmiddellijk als de semafoor momenteel 'bezet' is.

Bezit en vrijgave

Wanneer een semafoor wordt genomen, wordt gezegd dat de taak deze 'bezit.' Bij het 'vrijgeven' van een semafoor is geen time-out argument nodig. De vrijgave van een bezette semafoor vindt onmiddellijk plaats. Het 'vrijgeven' van een semafoor betekent niet noodzakelijkerwijs dat deze onmiddellijk door een andere taak zal worden 'bezet.'

Begintoestand

Een binaire semafoor wordt bij FreeRTOS in eerste instantie in de toestand 'bezet' aangemaakt. Dit is anders dan bij een mutex in Linux of Windows waar u een semafoor misschien mee vergelijkt, omdat een mutex in de 'vrijgegeven' wordt aangemaakt. In een volgende aflevering komen we terug op de mutex in FreeRTOS.

xSemaphoreBinaryCreate()

Een binaire semafoor aanmaken in FreeRTOS is niet moeilijk:

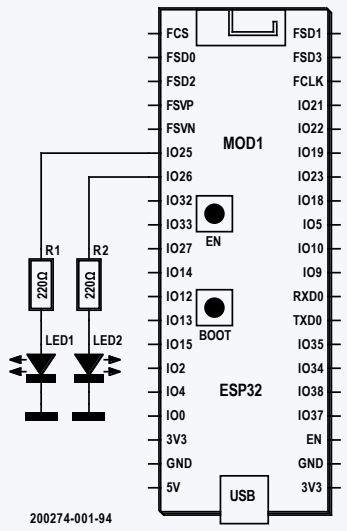
```
SemaphoreHandle_t hMySemaphore;

hMySemaphore = xSemaphoreCreateBinary();
assert(hMySemaphore)
```

Er hoeven geen argumenten te worden meegegeven; er wordt een *handle* naar de aangemaakte binaire semafoor geretourneerd. Het is mogelijk dat de handle wordt geretourneerd als `nullptr` (`NULL`) als u het beschikbare geheugen al hebt opgebruikt. Het verdient aanbeveling om de geretourneerde waarde (hier is de `assert()` macro van assert.h gebruikt) te controleren om er zeker van te zijn dat de semafoor is aangemaakt.

xSemaphoreGive()

Een semafoor vrijgeven is ook simpel:

```
SemaphoreHandle_t hMySemaphore;
BaseType_t rc; // return code

...
rc = xSemaphoreGive(hMySemaphore);
assert(rc == pdPASS);
```



Figuur 1. Aansluiten van de LED's voor de semaphores.ino-demo.

Het vrijgeven (give) kan alleen om de volgende redenen mislukken (dan wordt pdFAIL geretourneerd):

- de handle (hMySemaphore) is ongeldig;
- de semafore is al 'vergeven.'

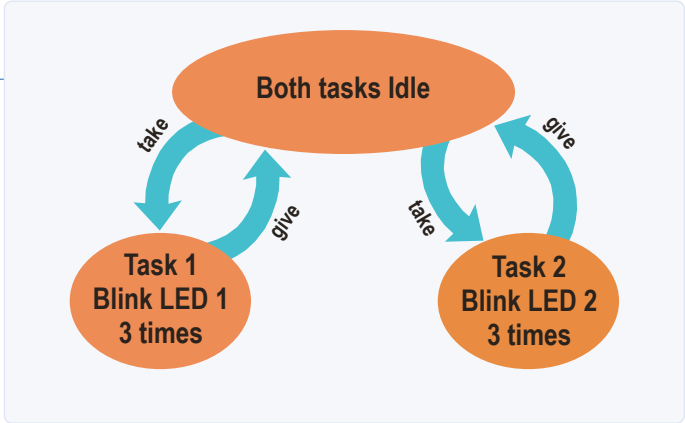
xSemaphoreTake()

Het nemen (take) van een semafoor is eventueel een blokkerende operatie voor de aanroepende taak:

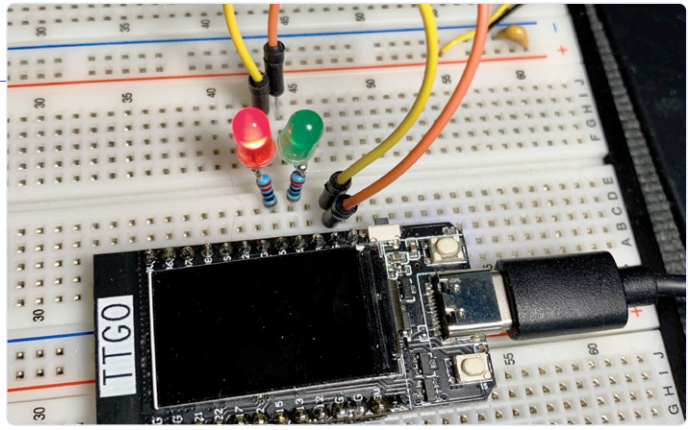
```
SemaphoreHandle_t hMySemaphore;
TickType_t wait = 30; // ticks
BaseType_t rc; // return code

...
rc = xSemaphoreTake(hMySemaphore, wait);
// Returns pdPASS when taken,
// otherwise returns pdFAIL if it times out
```

LISTING 1. HET DEMOPROGRAMMA SEMAPHORES.INO [1].

```
0001: // semaphores.ino
0002: // Practical ESP32 Multitasking
0003: // Binary Semaphores
0004:
0005: #define LED1_GPIO   25
0006: #define LED2_GPIO   26
0007:
0008: static SemaphoreHandle_t hsem;
0009:
0010: void led_task(void *argp) {
0011:   int led = (int)argp;
0012:   BaseType_t rc;
0013:
0014:   pinMode(led,OUTPUT);
0015:   digitalWrite(led,0);
0016:
0017:   for (;;) {
0018:     // First gain control of hsem
0019:     rc = xSemaphoreTake(hsem,portMAX_DELAY);
0020:     assert(rc == pdPASS);
0021:
0022:     for ( int x=0; x<6; ++x ) {
0023:       digitalWrite(led,digitalRead(led)^1);
0024:       delay(500);
0025:     }
0026:
0027:     rc = xSemaphoreGive(hsem);
0028:     assert(rc == pdPASS);
0029:   }
0030: }
0031:
0032: void setup() {
0033:   int app_cpu = xPortGetCoreID();
0034:   BaseType_t rc;  // Return code
0035:
0036:   hsem = xSemaphoreCreateBinary();
0037:   assert(hsem);
0038:
0039:   rc = xTaskCreatePinnedToCore(
0040:     led_task,   // Function
0041:     "led1task", // Task name
0042:     3000,       // Stack size
0043:     (void*)LED1_GPIO, // arg
0044:     1,          // Priority
0045:     nullptr,    // No handle returned
0046:     app_cpu);   // CPU
0047:   assert(rc == pdPASS);
0048:
0049:   // Allow led1task to start first
0050:   rc = xSemaphoreGive(hsem);
0051:   assert(rc == pdPASS);
0052:
0053:   rc = xTaskCreatePinnedToCore(
0054:     led_task,   // Function
0055:     "led2task", // Task name
0056:     3000,       // Stack size
0057:     (void*)LED2_GPIO, // argument
0058:     1,          // Priority
0059:     nullptr,    // No handle returned
0060:     app_cpu);   // CPU
0061:   assert(rc == pdPASS);
0062: }
0063:
0064: // Not used
0065: void loop() {
0066:   vTaskDelete(nullptr);
0067: }
```



Figuur 2. Status van de uitgevoerde taken in het programma semaphores.ino.



Figuur 3. De ESP32 TTGO stuurt twee LED's aan met het programma semaphores.ino.

Het 'nemen' kan mislukken als de handle ongeldig is of als er een time-out is opgetreden voor de aanroep (zoals gedefinieerd door het `wait`-argument). Anders zal de oproep de uitvoering van de aanroepende taak blokkeren totdat deze de semafoor heeft 'genomen'.
De `wait`-parameter kan één van drie verschillende waarden hebben, afhankelijk van wat de aanroeper wil:

- macro-waarde `portMAX_DELAY` – wacht eeuwig tot 'nemen' lukt;
- een positieve waarde ongelijk nul – wacht gedurende het opgegeven aantal ticks;
- zero – mislukt onmiddellijk als de semafoor niet onmiddellijk kan worden 'genomen'.

Demo

Als demonstratie laten we twee taken elk hun eigen LED knipperen (**listing 1**). Door een binaire semafoor te delen, kan telkens slechts één van de taken op elk moment zijn LED laten knipperen. De geblokkeerde taak moet wachten tot de semafoor is 'vrijgegeven' door de andere (lopende) taak, waardoor de semafoor weer beschikbaar komt om te worden 'genomen'. **Figuur 1** laat zien hoe de LED's worden aangesloten op welke ESP32 dan ook die u wenst te gebruiken.

Figuur 2 illustreert de toestanden van het tweetal taken dat wordt uitgevoerd in het demoprogramma. Wanneer de ene taak de semafoor in bezit neemt, kan die als bezitter zijn taak uitvoeren en de LED laten knipperen, terwijl de andere taak is geblokkeerd en moet wachten. Alleen wanneer de eigenaar de semafoor weer vrijgeeft, kan de andere taak deze in bezit nemen en worden uitgevoerd.
In **figuur 3** ziet u hoe we de schakeling op breadboard hebben opgebouwd met een TTGO ESP32 ontwikkelboard.

In de `setup()`-routine creëert regel 36 de semafoor en wijst de handle toe aan de statische globale variabele `hsem`. Regels 39 tot 46 maken de eerste taak om LED1 te laten knipperen (let op het argument in regel 43 van de aanroep voor het aanmaken van taken). De te gebruiken GPIO-pin wordt doorgegeven als een `void` pointer. Dit wordt dan teruggestuurd naar een `int` GPIO-waarde in de taakfunctie `led_task()` (regel 11) van de taakfunctie. We misbruiken de pointer om een waarde door te geven, maar dit werkt zolang de waarde in hetzelfde aantal bits past als een pointeradres.
Nadat de eerste taak is gemaakt, geven we de semafoor vrij in regel 50. Door dit te doen voordat de tweede taak wordt gemaakt, is het waarschijnlijk dat de eerste taak de semafoor als eerste in bezit krijgt. Regels 53 tot 60 creëren vervolgens de tweede taak. Beide taken voeren dezelfde functiecode `led_task()` uit, maar met verschillende argumenten die de te gebruiken LED GPIO-pin specificeren.
Regels 14 en 15 configureren de LED die binnen de taak wordt gebruikt. De taak komt dan in een eindeloze lus beginnend bij regel 17. De eerste stap die binnen deze lus wordt uitgevoerd, is het nemen van de semafoor (regel 19). Daar slaagt slechts één taak tegelijk in.
De taak die erin slaagt de semafoor te nemen, gaat in de lus verder met de regels 22 tot en met 25. Hier knippert de bij de taak horende LED driemaal. Als dat gedaan is, wordt de semafoor in regel 27 'gegeven', zodat de andere taak de semafoor kan 'nemen'. Op deze manier geeft elke taak om de beurt de controle over aan de andere taak.

Samenvattend

De binaire semafoor die we hier gebruikten, werd op twee verschillende manieren benut. Vóór de eerste 'give'-bewerking in de functie `setup()` kan geen van beide taken doorgaan. Hierdoor gedroeg de semafoor zich als een blokkade omdat geen enkele taak kon worden voortgezet. Na die eerste 'give' -bewerking, 'neemt' een van de twee taken de semafoor, laat zijn LED knipperen en 'geeft' de semafoor terug. Op deze manier gebruikt lijkt de semafoor zich te gedragen als een mutex. Beide taken proberen voortdurend hun code uit te voeren, maar door de wederzijdse uitsluiting van de semafoor mag slechts één taak zijn LED laten knipperen.
Het is belangrijk om te onthouden dat de binaire semaforen worden aangemaakt in de toestand 'taken' (in tegenstelling tot een mutex). Als de semafoor in deze demo aanvankelijk niet was vrijgegeven in de `setup()`-routine (regel 50), zouden beide taken voor altijd vastzitten. Er zijn binnen FreeRTOS andere synchronisatieprimitieven beschikbaar (inclusief de mutex), maar soms is de eenvoudige binaire semafoor alles wat nodig is.

200274-04

WEBLINK

[1] https://github.com/ve3wwg/esp32_freertos/blob/master/semaphores/semaphores.ino